УДК 004.04

Мамонтов В.Д.
студент
2 курс магистратуры,
факультет «Системы управления, информатика и электроэнергетика»
Московский авиационный нститут
Россия, г. Москва

РЕАЛИЗАЦИЯ КОДА ХАФФМАНА НА ЯЗЫКЕ GO

Аннотация: *В данной статье описывается реализация алгоритма кодирования Хаффмана на языке программирования Go. Рассматривается принцип работы алгоритма, его применение и преимущества. Также представляется реализация алгоритма на языке Go с использованием стандартной библиотеки.*

Ключевые слова: *Алгоритм Хаффмана, кодирование, декодирование, оптимальное префиксное кодирование, Go.*

Annotation: *This article describes the implementation of the Huffman coding algorithm in the Go programming language. The principle of operation of the algorithm, its application and advantages are considered. The implementation of the algorithm in the Go language using the standard library is also presented.*

Key words: *Huffman algorithm, encoding, decoding, optimal prefix encoding, Go.*

Код Хаффмана - это метод оптимального префиксного кодирования, который используется для сжатия данных. Его основная идея заключается в присвоении переменной длины кодов каждому символу в зависимости от его частоты встречаемости в исходном тексте. Таким образом, часто

встречающиеся символы получают более короткие коды, что повышает общую эффективность сжатия.

Пример:

Пусть у нас есть текст "abracadabra". В этом случае буква "a" встречается 5 раз, "b" - 2 раза, "r" - 2 раза, "c" - 1 раз, "d" - 1 раз. С использованием кода Хаффмана мы можем закодировать "a" как "0", "b" как "101", "r" как "100", "c" как "1100" и "d" как "1101". После этого закодированный текст будет занимать меньше места по сравнению с исходным текстом.

Алгоритм Хаффмана актуален в сфере сжатия данных, поскольку он позволяет значительно уменьшить объем передаваемой информации, что особенно важно в условиях ограниченной пропускной способности сети. Его применение также распространено в сфере хранения и передачи больших объемов данных.

Для реализации кода Хаффмана на языке Go используется стандартная библиотека, предоставляющая необходимые инструменты для работы с битовыми операциями и структурами данных. Создание алгоритма Хаффмана в Go включает в себя формирование дерева кодирования, построение кодов для символов и реализацию процедур кодирования и декодирования.

```
package main
import (
	"container/heap"
	"fmt"
)
// Node представляет узел дерева Хаффмана
type Node struct {
```

```
	char  rune
	freq  int
	left  *Node
	right *Node
}
// Priority queue представляет очередь с приоритетом для узлов дерева Хаффмана
type PriorityQueue []*Node
// Len возвращает длину очереди
func (pq PriorityQueue) Len() int { return len(pq) }
// Less определяет, меньше ли элемент с индексом i, элемента с индексом j
func (pq PriorityQueue) Less(i, j int) bool {
	return pq[i].freq < pq[j].freq
}
// Swap меняет местами элементы с индексами i и j
func (pq PriorityQueue) Swap(i, j int) {
	pq[i], pq[j] = pq[j], pq[i]
}
// Push добавляет элемент в очередь
func (pq *PriorityQueue) Push(x interface{}) {
	item := x.(*Node)
	*pq = append(*pq, item)
}
// Pop удаляет и возвращает последний элемент из очереди
func (pq *PriorityQueue) Pop() interface{} {
	old := *pq
	n := len(old)
	item := old[n-1]
	*pq = old[0 : n-1]
```

```
	return item
}
// BuildTree строит дерево Хаффмана на основе символов и их частот
func buildHuffmanTree(freqmap map[rune]int) *Node {
	pq := make(PriorityQueue, len(freqmap))
	i := 0
	for char, freq := range freqmap {
		pq[i] = &Node{char, freq, nil, nil}
		i++
	}
	heap.Init(&pq)
	for pq.Len() > 1 {
		min1 := heap.Pop(&pq).(*Node)
		min2 := heap.Pop(&pq).(*Node)

		newNode := &Node{0, min1.freq + min2.freq, min1, min2}
		heap.Push(&pq, newNode)
	}
	return heap.Pop(&pq).(*Node)
}

func printCodes(root *Node, prefix []byte) {
	if root != nil {
		if root.left == nil && root.right == nil {
			fmt.Printf("%c: %s\n", root.char, string(prefix))
		}
		printCodes(root.left, append(prefix, '0'))
		printCodes(root.right, append(prefix, '1'))
	}
```

```
}
func main() {
	input := "abracadabra"
	freqmap := make(map[rune]int)
	for _, char := range input {
		freqmap[char]++
	}
	root := buildHuffmanTree(freqmap)
	fmt.Println("Huffman Codes:")
	printCodes(root, []byte{})
}
```

Этот код демонстрирует простую реализацию алгоритма Хаффмана на языке Go. Входная строка "abracadabra" используется для построения дерева Хаффмана, а затем выводятся коды для каждого символа на основе полученного дерева.

Реализация алгоритма Хаффмана на языке программирования Go позволяет использовать его мощные возможности для сжатия и передачи информации. Это обеспечивает эффективное управление данными, что является ключевым аспектом в мире современных информационных технологий.

Использованные источники:

1. Тим Рафгарден. Совершенный алгоритм Жадные алгоритмы и динамическое программирование. — СПб.: Питер, 2020. — 256 с.: ил. — (Серия «Библиотека программиста»).

2. Alan A. A. Donovan, Brian W. Kernighan. "The Go Programming Language" — СПб.: Питер, 2016. — 1338 с.: ил.

3. Чернышев С. А. Программирование на языке Go —КноРус.: Москва, 2023. — 755 с.: ил.